Finding Similar Items: Locality Sensitive Hashing

CS246: Mining Massive Datasets

Jure Leskovec, Stanford University

http://cs246.stanford.edu

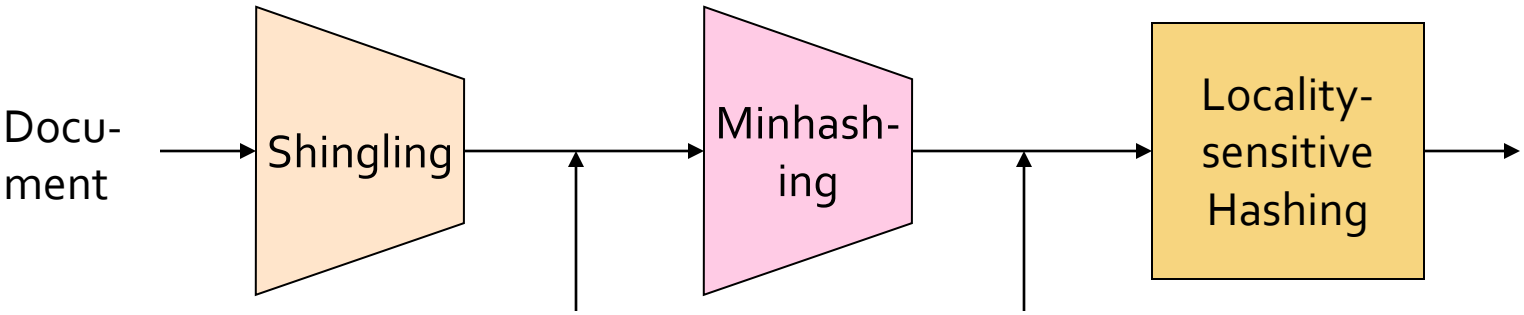
Recap: Finding similar documents

- Goal: Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates"
- Application:
  - Detect mirror and approximate mirror sites/pages:
    - Don't want to show both in a web search
- Problems:
  - Many small pieces of one doc can appear out of order in another
  - Too many docs to compare all pairs
  - Docs are so large or so many that they cannot fit in main memory

3 Essential Steps for Similar Docs

1. *Shingling*: Convert documents to large sets of items
2. *Minhashing*: Convert large sets into short signatures, while preserving similarity
3. *Locality-sensitive hashing*: Focus on pairs of signatures likely to be from similar documents

The Big Picture



Candidate pairs : those pairs of signatures that we need to test for similarity.

The set of strings of length *k* that appear in the doc-ument

Signatures : short integer vectors that represent the sets, and reflect their similarity

Recap: Shingles

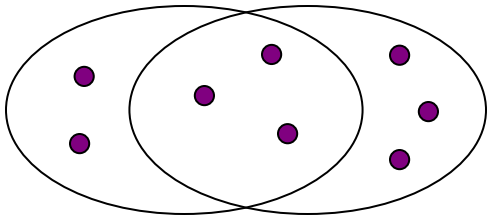
- A *k*-shingle (or *k*-gram) for a document is a sequence of *k* tokens that appears in the document
  - Tokens can be characters, words or something else, depending on application
  - Assume tokens = characters for examples
- Example: k=2; D_1= abcab
  Set of 2-shingles: $S(D_1)$={ab, bc, ca}
- Represent a doc by the set of hash values of its *k*-shingles

Similarity Metric for Shingles

- Document D_1 = set of k-shingles $C_1=S(D_1)$
- Equivalently, each document is a 0/1 vector in the space of k-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse
- A natural similarity measure is the Jaccard similarity:

$$Sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$

Encoding sets as bit vectors



- We can encode sets using 0/1 (bit, boolean) vectors
  - One dimension per element in the universal set
- Interpret set intersection as bitwise AND, and set union as bitwise OR
- Example: C_1 = 1100011; C_2 = 0110010
  - Size of intersection = 2; size of union = 5, Jaccard similarity (not distance) = 2/5
  - $d(C_1,C_2)$ = 1 – (Jaccard similarity) = 3/5

shingles

1	0	1	0
1	1	0	1
0	1	0	1
0	0	0	1
0	0	0	1
1	1	1	0
1	0	1	0

documents

Outline: Finding Similar Columns

1. Signatures of columns = small summaries of columns
2. Examine pairs of signatures to find similar signatures
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

- Warnings:
  1. Comparing all pairs of signatures may take too much time, even if not too much space
     - A job for Locality-Sensitive Hashing
  2. These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Singatures)

- Key idea: "hash" each column *C* to a small *signature* h(C), such that:
  1. h(C) is small enough that we can fit a signature in main memory for each column
  2. $Sim(C_1, C_2)$ is the same as the "similarity" of $h(C_1)$ and $h(C_2)$
- Goal: Find a hash function *h()* such that:
  - if $Sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $Sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$
- Hash docs into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket

Min-hashing

- Clearly, the hash function depends on the similarity metric
  - Not all similarity metrics have a suitable hash function
- There is a suitable hash function for Jaccard similarity
  - **Min-hashing**

Min-hashing

- Imagine the rows of the boolean matrix permuted under random permutation π
- Define a "hash" function $h_\pi(C)$ = the number of the first (in the permuted order π) row in which column C has 1:

$$h_\pi(C)=\min \pi(C)$$

- Use several (e.g., 100) independent hash functions to create a signature

Minhashing Example

Permutation π

1	4	3
3	2	4
7	1	7
6	3	6
2	6	1
5	7	2
4	5	5

Input matrix

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Surprising Property

- Choose a random permutation π
- Prob. that $h_\pi(C_1) = h_\pi(C_2)$ is the same as $Sim(C_1, C_2)$:

  $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = Sim(C_1, C_2)$
- Why?
  - Let X be a set of shingles, $X \subseteq [2^{64}]$, $x \in X$
  - Then: $\Pr[\pi(x) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $x \in X$ is mapped to the min element
  - Let x be s.t. $\pi(x) = \min(\pi(C_1 \cup C_2))$
  - Then either: $\pi(x) = \min(\pi(C_1))$ if $x \in C_1$, or

    $\pi(x) = \min(\pi(C_2))$ if $x \in C_2$
  - So the prob. that both are true is the prob. $x \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2| = Sim(C_1, C_2)$

Four Types of Rows

- Given cols C_1 and C_2, rows may be classified as:

	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

- Also, a = # rows of type a , etc.
- **Note:** $Sim(C_1, C_2) = a/(a + b + c)$
- **Then: Pr**$[h(C_1) = h(C_2)] = Sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1
  - If it's a type-*a* row, then $h(C_1) = h(C_2)$
    If a type-*b* or type-*c* row, then not

Similarity for Signatures

- The *similarity of two signatures* is the fraction of the hash functions in which they agree
- Note: Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Min Hashing – Example

1	4	3
3	2	4
7	1	7
6	3	6
2	6	1
5	7	2
4	5	5

Input matrix

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

Minhash Signatures

- Pick (say) 100 random permutations of the rows
- Think of *Sig*(C) as a column vector
- Let *Sig*(C)[i] =

  according to the *i*-th permutation, the index of the first row that has a 1 in column *C*
- Note: We store the sketch of document C in ~100 bytes:

  $Sig(C)[i] = \min(\pi_i(C))$

Implementation – (1)

- Suppose the matrix has 1 billion rows
- Hard to pick a random permutation from 1…billion
- Representing a random permutation requires 1 billion entries
- Accessing rows in permuted order leads to thrashing

Implementation – (2)

- A good approximation to permuting rows: pick 100 (?) hash functions
  - $h_1, h_2, \ldots$
  - For rows r and s, if $h_i(r) < h_i(s)$, then r appears before s in permutation i.
- For each column c and each hash function h_i, keep a "slot" $M(i, c)$
- Intent: $M(i, c)$ will become the smallest value of $h_i(r)$ for which column c has 1 in row r
  - i.e., $h_i(r)$ gives order of rows for i-th permuation

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$

$h(1)=1,\ h(2)=2,\ h(3)=3,\ h(4)=4,\ h(5)=0$

$h(C1) = 1$

$h(C2) = 0$

$g(x) = 2x+1 \bmod 5$

$g(1)=3,\ g(2)=0,\ g(3)=2,\ g(4)=4,\ g(5)=1$

$g(C1) = 2$

$g(C2) = 0$

$Sig(C1) = [1,2]$

$Sig(C2) = [0,0]$

Implementation – (3)

- Sort the input matrix so it is ordered by rows
  - So can iterate by reading rows sequentially from disk

for each row r
  for each column c
    if c has 1 in row r
      for each hash function h_i **do**
        if $h_i(r) < M(i, c)$ **then**
          $M(i, c) := h_i(r)$

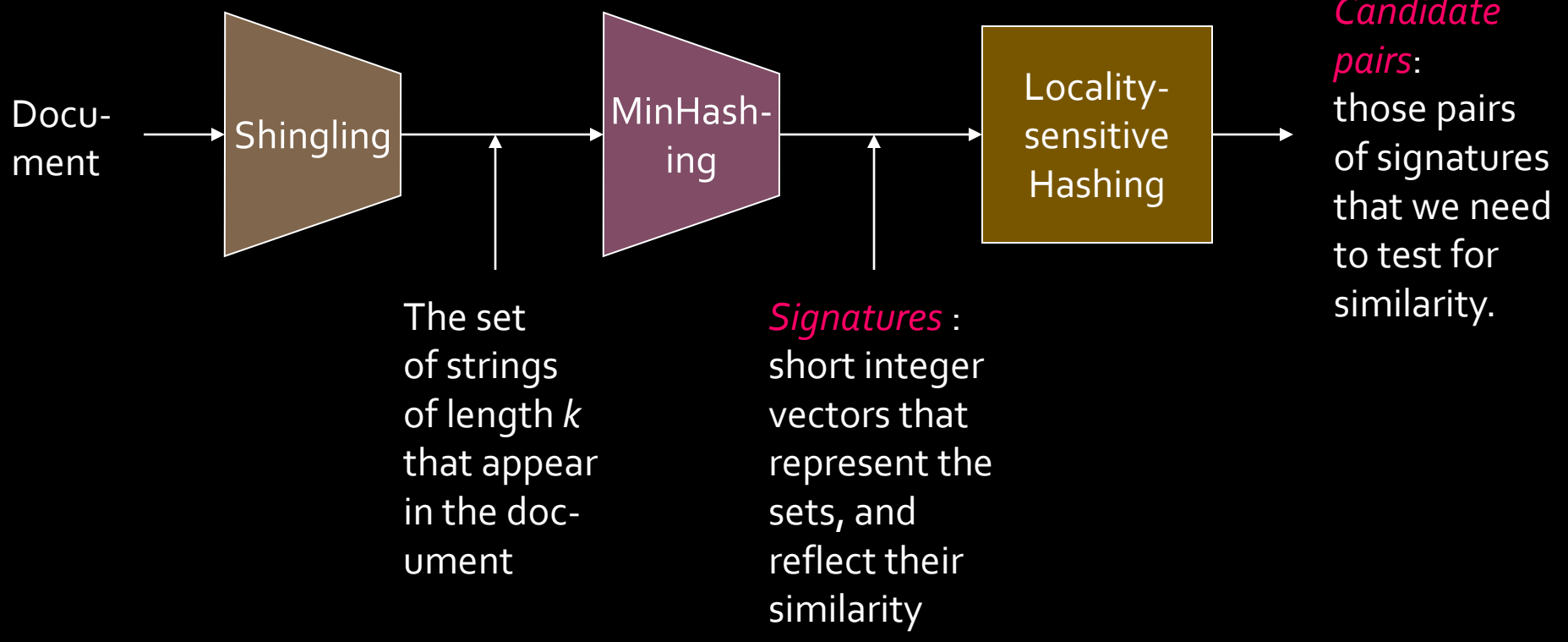
Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

$h(x) = x \bmod 5$
$g(x) = 2x+1 \bmod 5$

	Sig(C_1)	Sig2(C_2)
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

$M(i, c)$



Locality Sensitive Hashing

LSH: First Cut

- Goal: Pick a similarity threshold *s*, e.g., *s* = 0.8
  Find documents with Jaccard similarity at least *s*

- **LSH –** General idea: Use a function f(x,y) that tells whether or not *x* and *y* is a *candidate pair*: a pair of elements whose similarity must be evaluated
  - For minhash matrices: Hash columns to many buckets, and make elements of the same bucket candidate pairs
  - Each pair of documents that hashes into the same bucket is a candidate pair

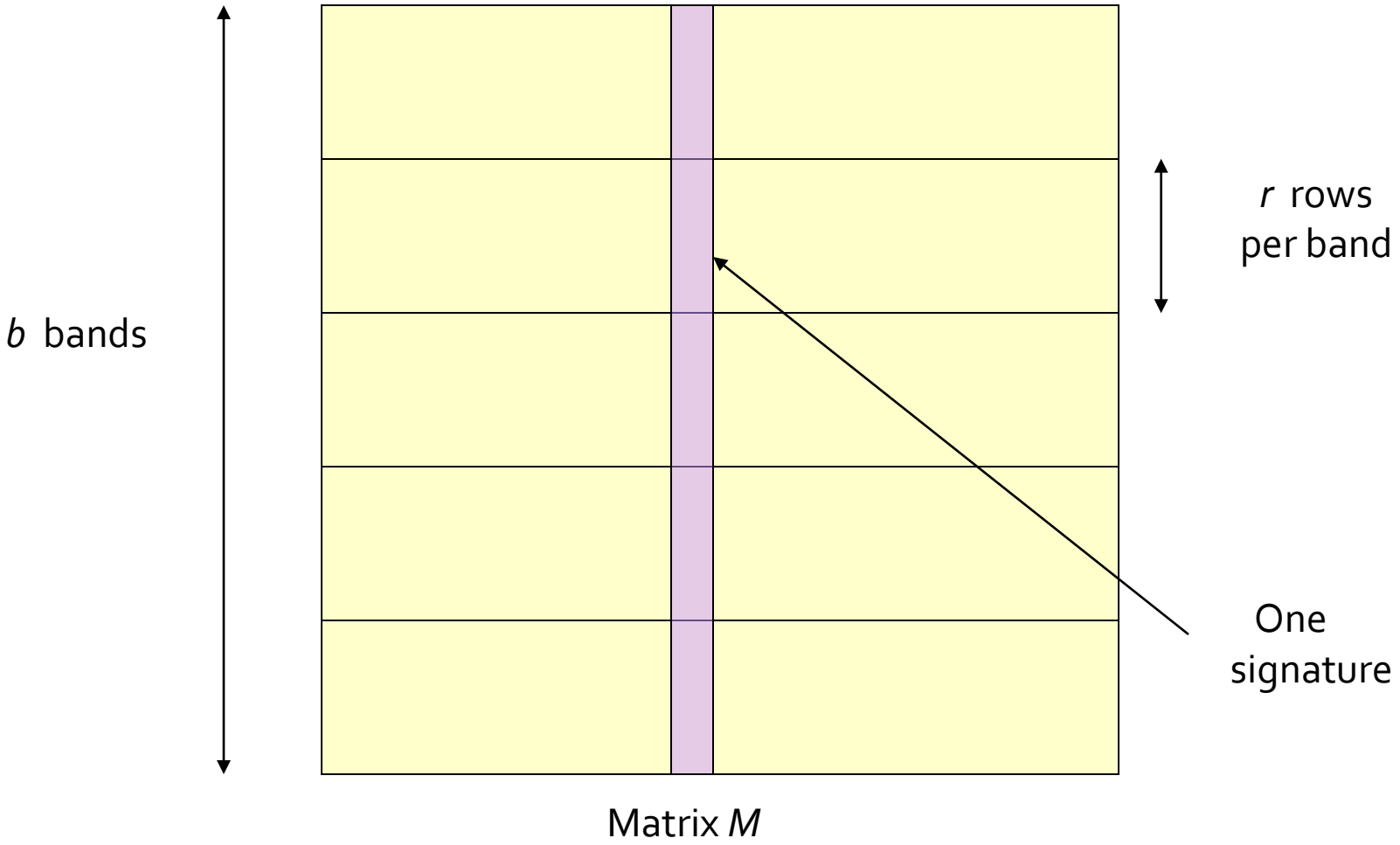
Candidates from Minhash signatures

- Pick a similarity threshold *s*, a fraction < 1
- Columns *x* and *y* are a candidate pair if their signatures agree on at least fraction *s* of their rows:
  $M(i, x) = M(i, y)$ for at least frac. *s* values of *i*
  - We expect documents *x* and y to have the same similarity as their signatures

LSH for Minhash signatures

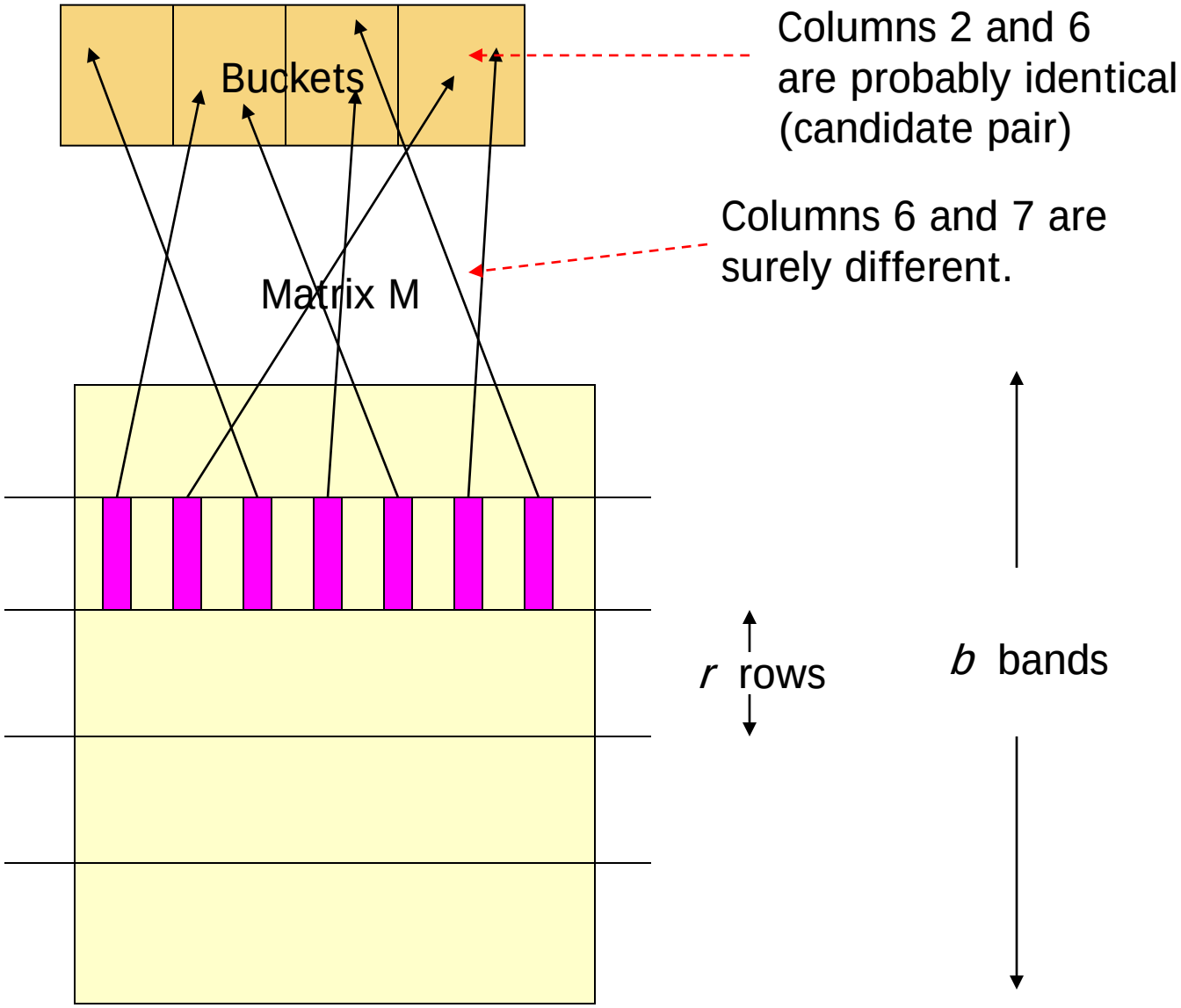
- Big idea: hash columns of signature matrix *M* several times.
- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability
- Candidate pairs are those that hash to the same bucket

Partition Into Bands



Partition into Bands – (2)

- Divide matrix *M* into *b* bands of *r* rows.
- For each band, hash its portion of each column to a hash table with *k* buckets.
  - Make *k* as large as possible.
- *Candidate* column pairs are those that hash to the same bucket for ≥ 1 band.
- Tune *b* and *r* to catch most similar pairs, but few nonsimilar pairs.


Buckets
Columns 2 and 6 are probably identical (candidate pair)
Columns 6 and 7 are surely different.
Matrix M
r rows
b bands

Simplifying Assumption

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band.
- Hereafter, we assume that "same bucket" means "identical in that band."
- Assumption needed only to simplify analysis, not for correctness of algorithm.

Example of bands

- Suppose 100,000 columns
- Signatures of 100 integers.
- Therefore, signatures take 40Mb.
- Choose 20 bands of 5 integers/band.

- Goal: find pairs of documents that are at least 80% similar.

Suppose C_1, C_2 are 80% Similar

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$.
- Probability C_1, C_2 are *not* similar in any of the 20 bands: $(1-0.328)^{20} = .00035$.
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives
  - We would find 99.965% pairs of truly similar documents

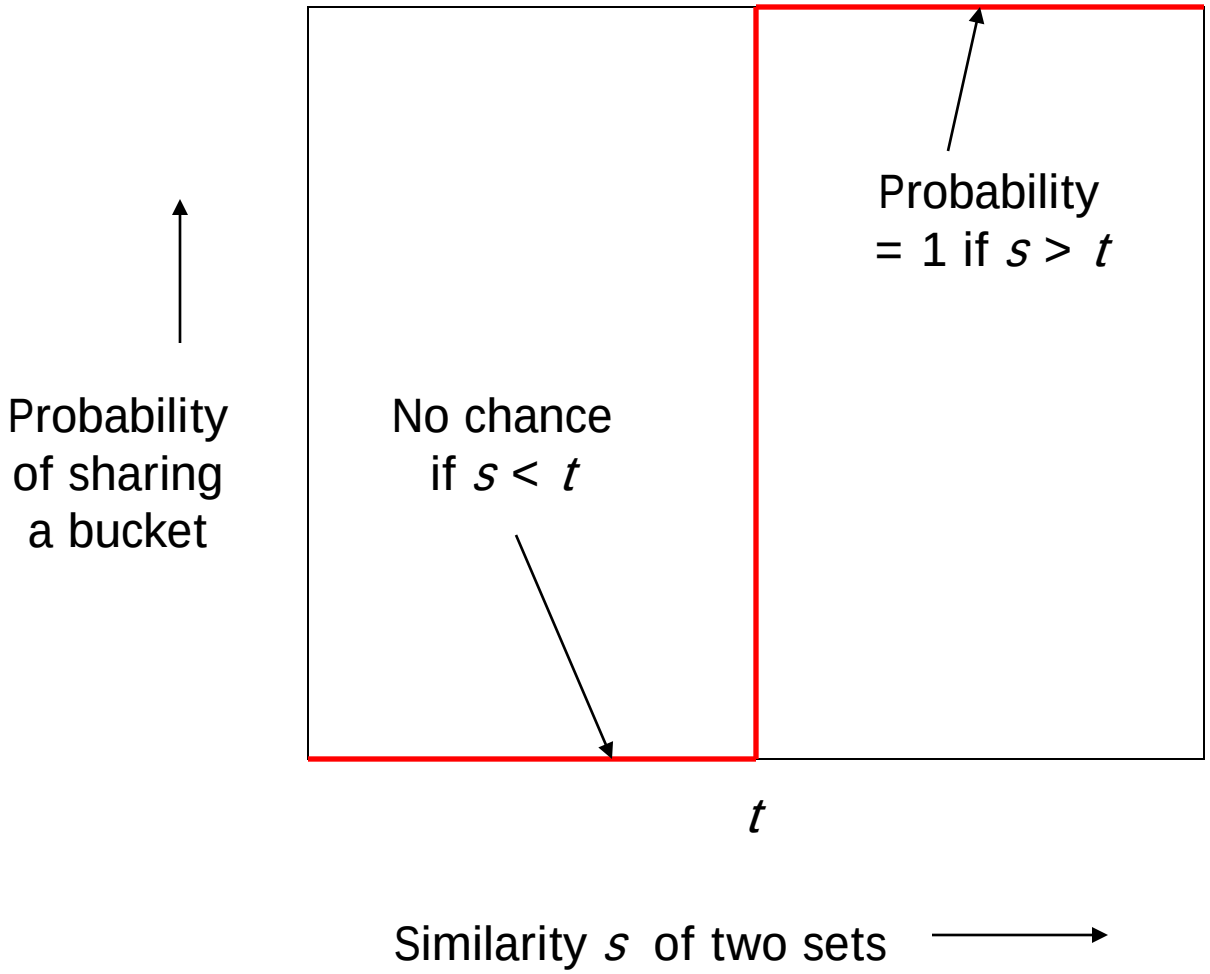
Suppose C_1, C_2 are only 30% similar

- Probability C_1, C_2 identical in any one particular band: $(0.3)^5 = 0.00243$
- Probability C_1, C_2 identical in $\geq$ 1 of 20 bands: $\leq 20 * 0.00243 = 0.0486$
- In other words, approximately 4.86% pairs of docs with similarity 30% end up becoming candidate pairs
  - False positives

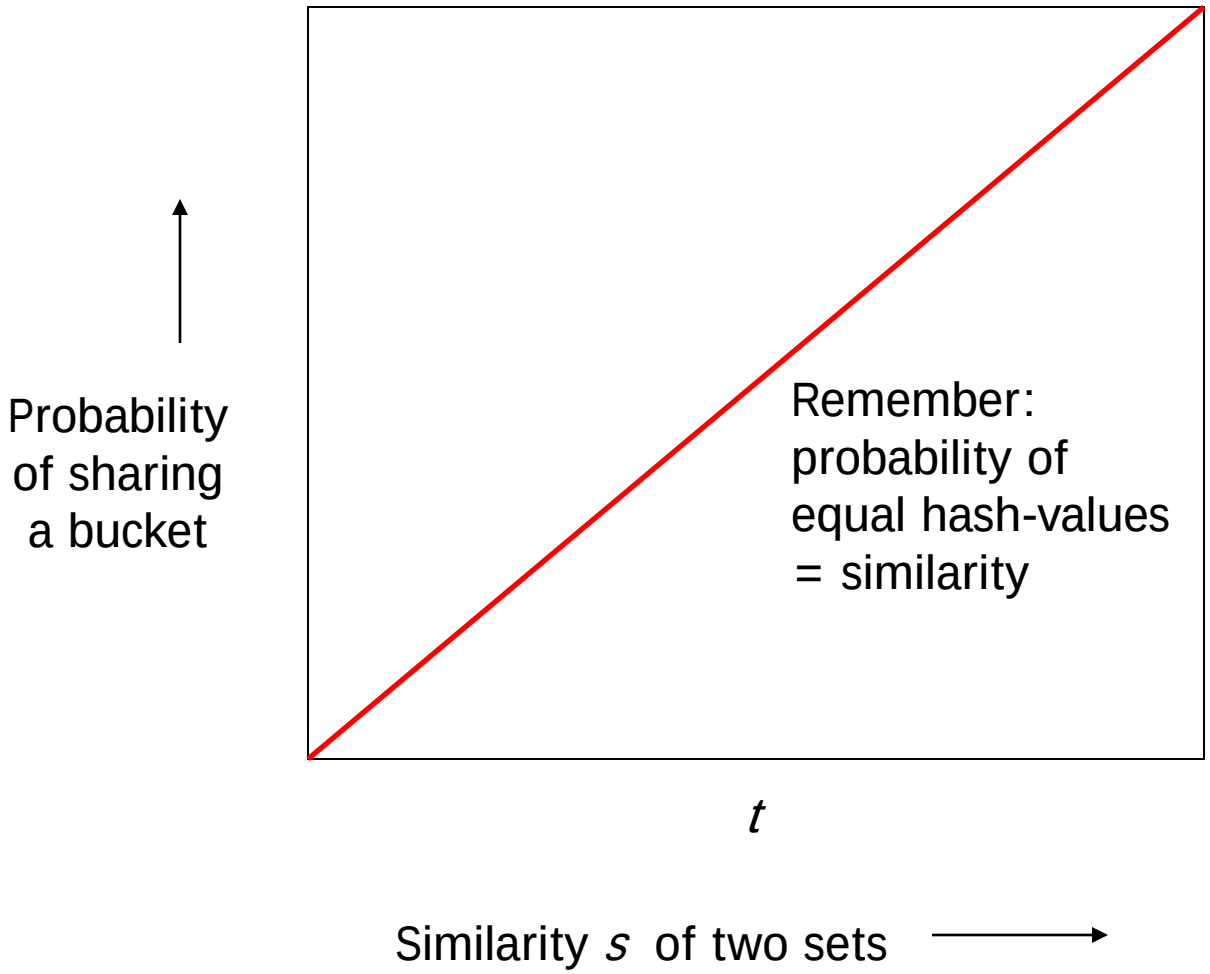
LSH Involves a Tradeoff

- Pick the number of minhashes, the number of bands, and the number of rows per band to balance false positives/negatives
- Example: if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up.

Analysis of LSH – What We Want



What One Band of One Row Gives You

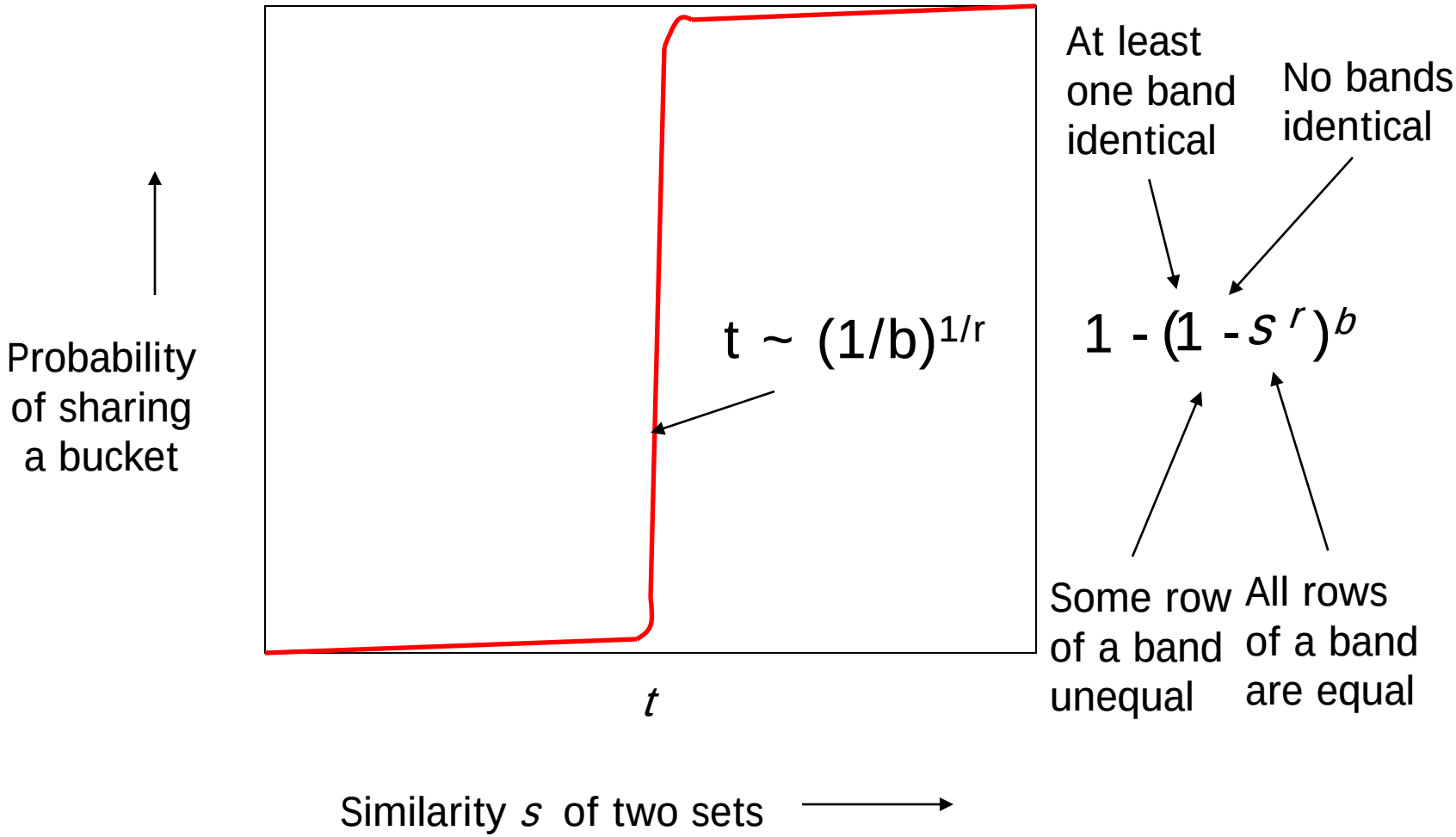


b bands, r rows/band

- Columns C and D have similarity s
- Pick any band (r rows)
  - Prob. that all rows in band equal = s^r
  - Prob. that some row in band unequal = $1 - s^r$
- Prob. that no band identical = $(1 - s^r)^b$
- Prob. that at least 1 band identical =

$$1 - (1 - s^r)^b$$

What *b* Bands of *r* Rows Gives You

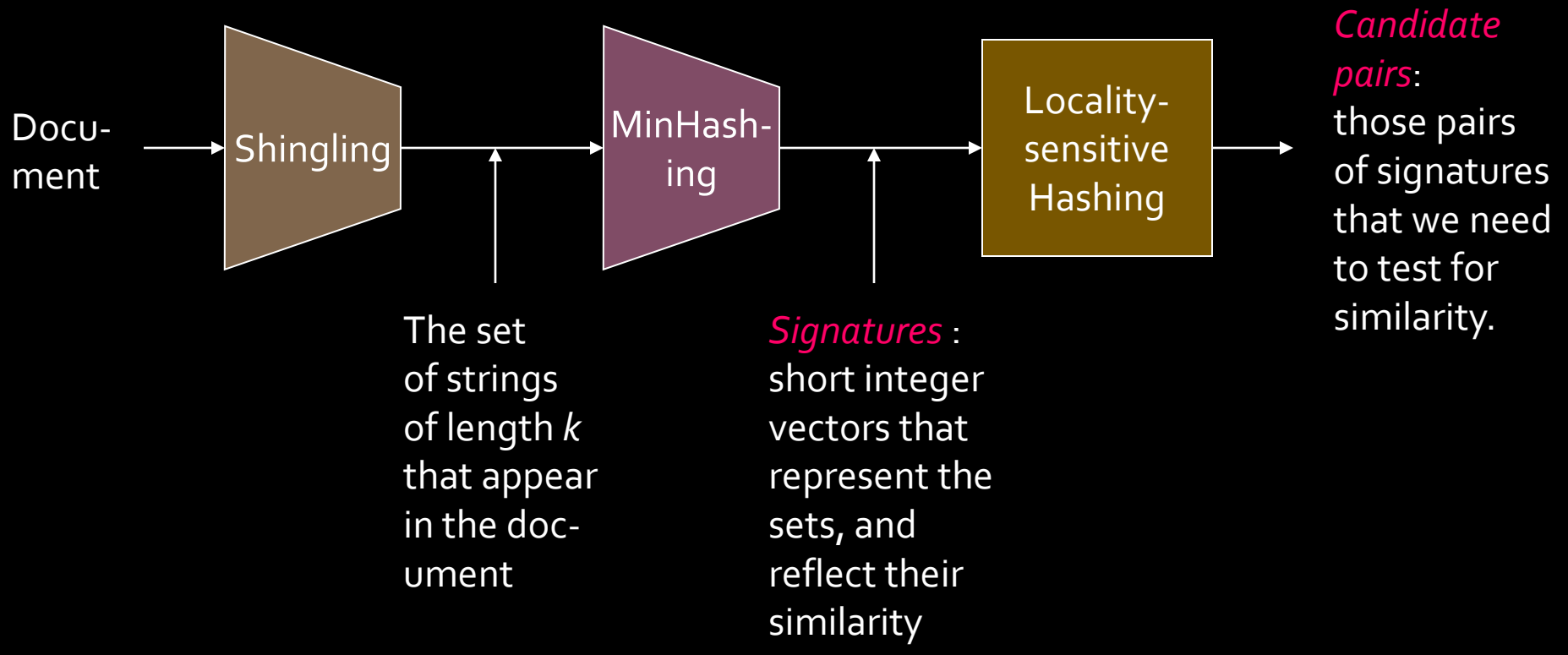


Example: $b = 20$; $r = 5$

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

LSH Summary

- Tune to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures.
- Check in main memory that candidate pairs really do have similar signatures.
- Optional: In another pass through data, check that the remaining candidate pairs really represent similar documents.



Theory of LSH

Theory of LSH

- We have used LSH to find similar documents
  - In reality, columns in large sparse matrices with high Jaccard similarity
  - e.g., customer/item purchase histories
- Can we use LSH for other distance measures?
  - e.g., Euclidean distances, Cosine distance
  - Let's generalize what we've learned!

Families of Hash Functions

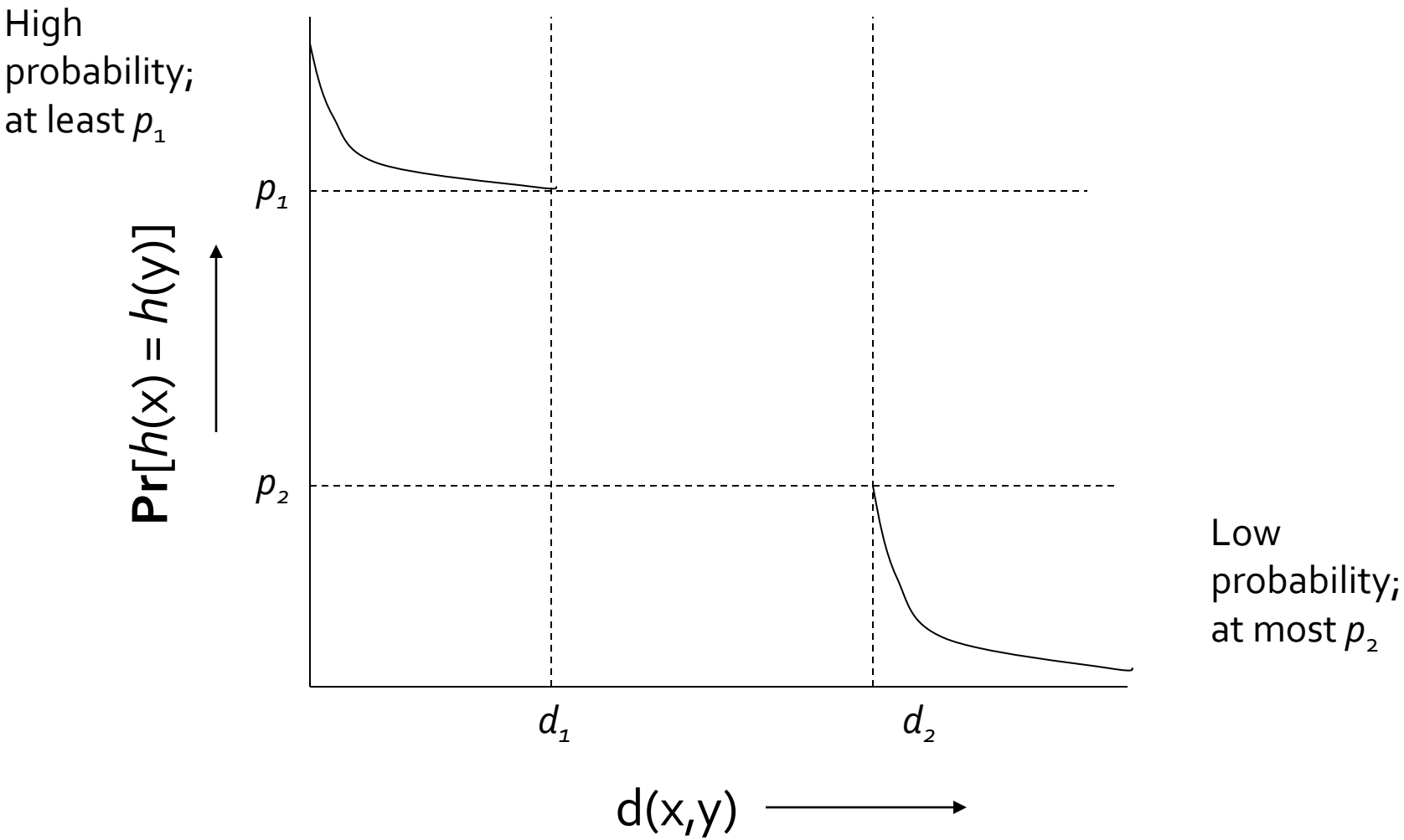
- For min-hash signatures, we got a min-hash function for each permutation of rows
- An example of a family of hash functions
  - A "hash function" is any function that takes *two* elements and says whether or not they are "equal" (really, are candidates for similarity checking).
    - Shorthand: h(x) = h(y) means "*h* says *x* and *y* are equal."
  - A *family* of hash functions is any set of hash functions
    - A set of related hash functions generated by some mechanism
  - We should be able to efficiently pick a hash function at random from such a family

Locality-Sensitive (LS) Families

- Suppose we have a space *S* of points with a distance measure *d*.

- A family **H** of hash functions is said to be *(d_1,d_2,p_1,p_2)-sensitive* if for any *x* and *y* in *S* :
  1. If $d(x,y) \leq d_1$, then prob. over all *h* in **H**, that $h(x) = h(y)$ is at least p_1.
  2. If $d(x,y) \geq d_2$, then prob. over all *h* in **H**, that $h(x) = h(y)$ is at most p_2.

A (d_1, d_2, p_1, p_2)-sensitive function



Example: LS Family

- Let S = sets, d = Jaccard distance, **H** is family of minhash functions for all permutations of rows
- Then for any hash function h in **H**,

  $$\mathbf{Pr}[h(x)=h(y)] = 1-d(x,y)$$

- Simply restates theorem about min-hashing in terms of distances rather than similarities

Example: LS Family – (2)

- Claim: **H** is a (1/3, 2/3, 2/3, 1/3)-sensitive family for *S* and *d*.

If distance $\leq$ 1/3
(so similarity $\geq$ 2/3)

Then probability
that minhash values
agree is $\geq$ 2/3

- For Jaccard similarity, minhashing gives us a $(d_1,d_2,(1-d_1),(1-d_2))$-sensitive family for any $d_1<d_2$
- Theory leaves unknown what happens to pairs that are at distance between d_1 and d_2
  - Consequence: no guarantees about fraction of false positives in that range

Amplifying a LS-Family

- Can we reproduce the "S-curve" effect we saw before for any LS family?
- The "bands" technique we learned for signature matrices carries over to this more general setting
- Two constructions:
  - AND construction like "rows in a band."
  - OR construction like "many bands."

AND of Hash Functions

- Given family **H**, construct family **H'** consisting of r functions from **H**.
- For $h = [h_1,\ldots,h_r]$ in **H'**, h(x)=h(y) if and only if $h_i(x)=h_i(y)$ for all i.
- Theorem: If **H** is (d_1,d_2,p_1,p_2)-sensitive, then **H'** is $(d_1,d_2,(p_1)^r,(p_2)^r)$-sensitive.
- Proof: Use fact that h_i's are independent.

OR of Hash Functions

- Given family **H**, construct family **H'** consisting of *b* functions from **H**.
- For $h = [h_1,\ldots,h_b]$ in **H'**, h(x)=h(y) if and only if $h_i(x)=h_i(y)$ for some *i*.
- Theorem: If **H** is (d_1,d_2,p_1,p_2)-sensitive, then **H'** is $(d_1,d_2,1-(1-p_1)^b,1-(1-p_2)^b)$-sensitive.

Effect of AND and OR Constructions

- AND makes all probabilities shrink, but by choosing *r* correctly, we can make the lower probability approach 0 while the higher does not.
- OR makes all probabilities grow, but by choosing *b* correctly, we can make the upper probability approach 1 while the lower does not.

Composing Constructions

- *r*-way AND construction followed by *b*-way OR construction
  - Exactly what we did with minhashing
- Take points x and y s.t. **Pr**[*h*(x) = *h*(y)] = p
  - **H** will make (x,y) a candidate pair with prob. P
- Construction makes (x,y) a candidate pair with probability $1-(1-p^r)^b$
  - The S-Curve!
- Example: Take **H** and construct **H'** by the AND construction with *r* = 4. Then, from **H'**, construct **H''** by the OR construction with *b* = 4

Table for Function $1-(1-p^4)^4$

p	$1-(1-p^4)^4$
.2	.0064
.3	.0320
.4	.0985
.5	.2275
.6	.4260
.7	.6666
.8	.8785
.9	.9860

Example: Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.8785,.0064)-sensitive family.

OR-AND Composition

- Apply a b-way OR construction followed by an r-way AND construction
- Tranforms probability *p* into $(1-(1-p)^b)^r$.
  - The same S-curve, mirrored horizontally and vertically.
- Example: Take **H** and construct **H'** by the OR construction with *b* = 4. Then, from **H'**, construct **H''** by the AND construction with *r* = 4.

Table for Function $(1-(1-p)^4)^4$

p	$(1-(1-p)^4)^4$
.1	.0140
.2	.1215
.3	.3334
.4	.5740
.5	.7725
.6	.9015
.7	.9680
.8	.9936

Example: Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.9936,.1215)-sensitive family.

Cascading Constructions

- Example: Apply the (4,4) OR-AND construction followed by the (4,4) AND-OR construction.
- Transforms a (.2,.8,.8,.2)-sensitive family into a (.2,.8,.9999996,.0008715)-sensitive family.
- Note this family uses 256 of the original hash functions.

Summary

- Pick any two distances $x < y$
- Start with a (x, y, (1-x), (1-y))-sensitive family
- Apply constructions to produce (x, y, p, q)-sensitive family, where p is almost 1 and q is almost 0.
- The closer to 0 and 1 we get, the more hash functions must be used.

LSH for Cosine Distance

- For cosine distance, there is a technique called Random Hyperplanes
  - Technique similar to minhashing
- A $(d_1, d_2, (1-d_1/180), (1-d_2/180))$-sensitive family for any d_1 and d_2.

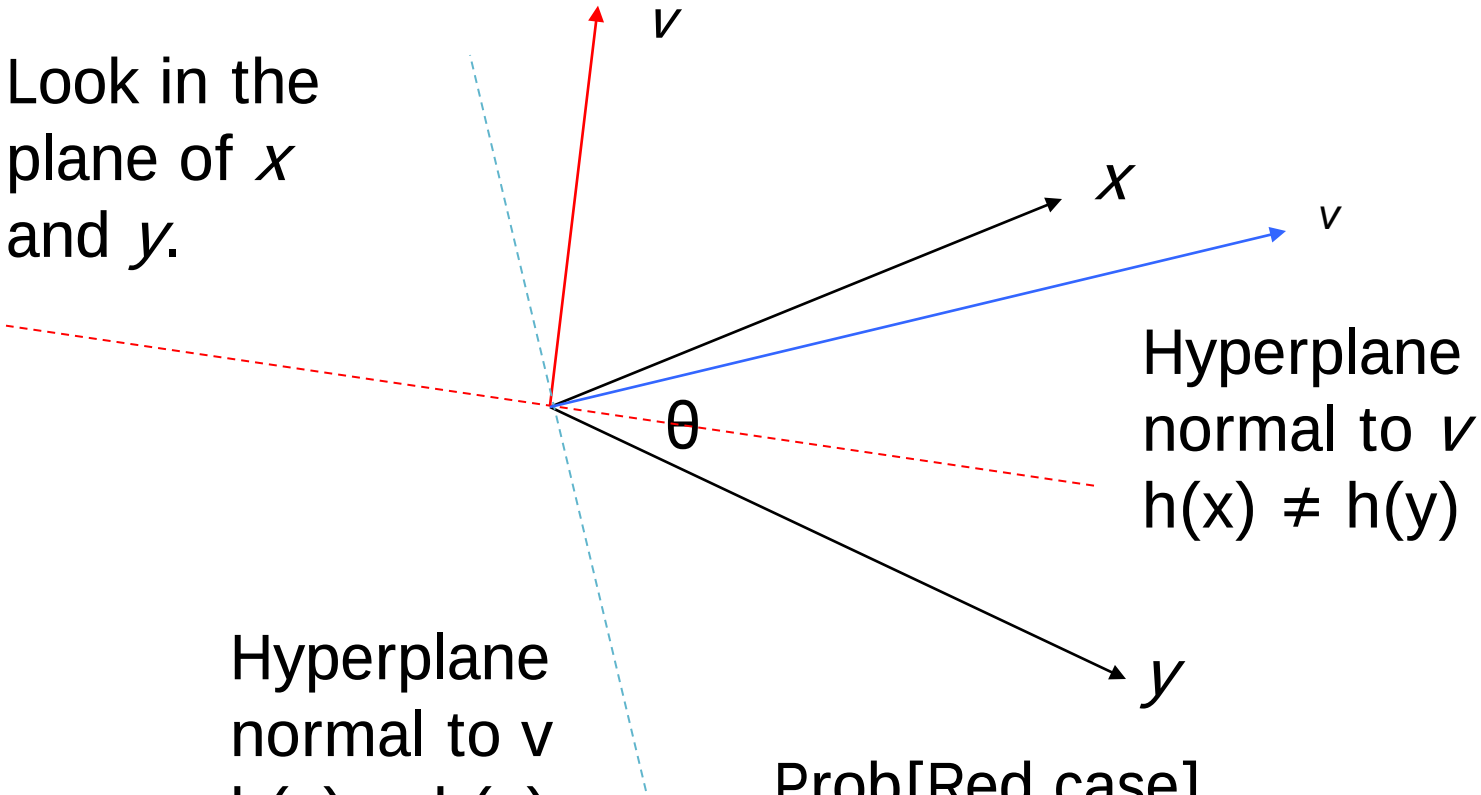
Random Hyperplanes

- Pick a random vector *v*, which determines a hash function h_v with two buckets.
- $h_v(x) = +1$ if $v.x > 0$; $= -1$ if $v.x < 0$.
- LS-family **H** = set of all functions derived from any vector.
- Claim: For points x and y,

  $\mathbf{Pr}[h(x)=h(y)] = 1 - d(x,y)/180$

Proof of Claim

Look in the plane of *x* and *y*.



Hyperplane normal to *v*
h(x) ≠ h(y)

Hyperplane normal to v
h(x) = h(y)

Prob[Red case] = θ/180

Signatures for Cosine Distance

- Pick some number of random vectors, and hash your data for each vector.
- The result is a signature (*sketch*) of +1's and –1's for each data point
- Can be used for LSH like the minhash signatures for Jaccard distance.
- Amplified using AND and OR constructions

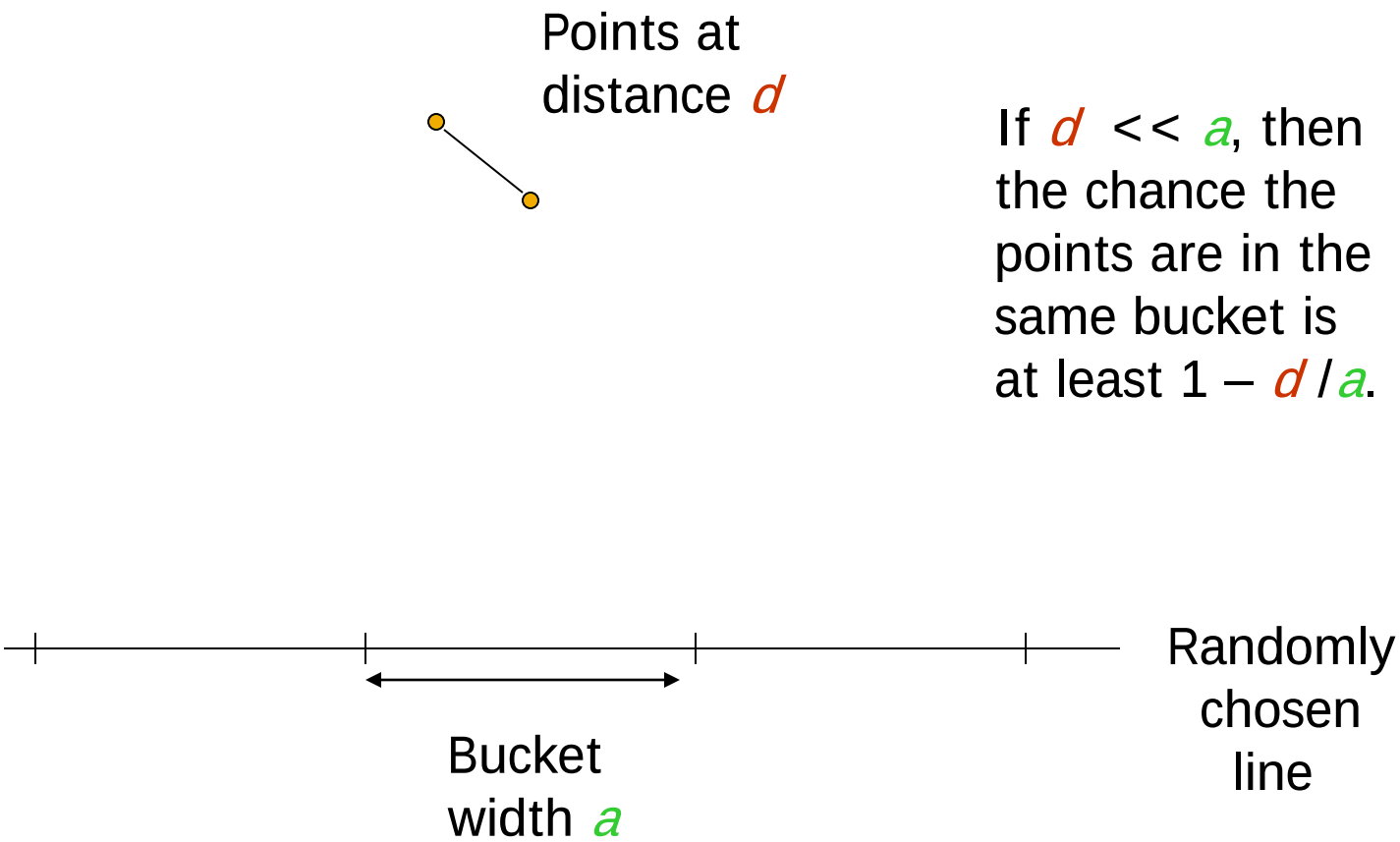
How to pick random vectors

- Expensive to pick a random vector in M dimensions for large M
  - M random numbers
- A more efficient approach
  - It suffices to consider only vectors *v* consisting of +1 and –1 components.
  - Why is this more efficient?

LSH for Euclidean Distance

- Simple idea: hash functions correspond to lines.
- Partition the line into buckets of size a.
- Hash each point to the bucket containing its projection onto the line.
- Nearby points are always close; distant points are rarely in same bucket.

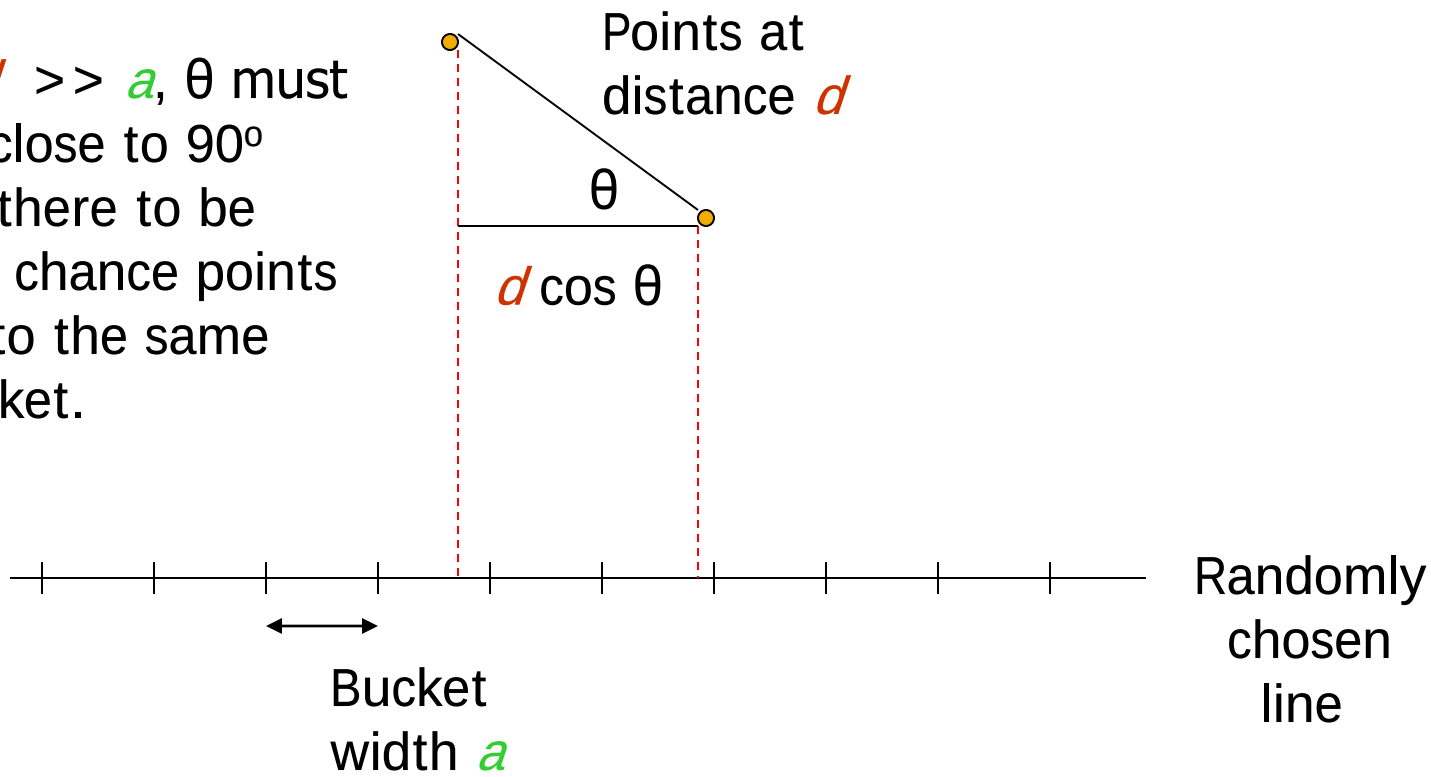
Projection of Points



If $d << a$, then the chance the points are in the same bucket is at least $1 - d/a$.

Projection of Points

If d >> a, θ must be close to 90° for there to be any chance points go to the same bucket.



An LS-Family for Euclidean Distance

- If points are distance $d \leq a/2$, prob. they are in same bucket $\geq 1 - d/a = 1/2$
- If points are distance $\geq 2a$ apart, then they can be in the same bucket only if $d \cos \theta \leq a$
  - $\cos \theta \leq ½$
  - $60 \leq \theta \leq 90$
  - I.e., at most 1/3 probability.
- Yields a $(a/2, 2a, 1/2, 1/3)$-sensitive family of hash functions for any a.
- Amplify using AND-OR cascades

Fixup: Euclidean Distance

- For previous distance measures, we could start with an (x, y, p, q)-sensitive family for any $x < y$, and drive p and q to 1 and 0 by AND/OR constructions.
- Here, we seem to need $y \geq 4x$.

Fixup – (2)

- But as long as $x < y$, the probability of points at distance x falling in the same bucket is greater than the probability of points at distance y doing so.
- Thus, the hash family formed by projecting onto lines is an (x, y, p, q)-sensitive family for some $p > q$.
  - Then, amplify by AND/OR constructions.